2 Let Envelope-Recipient = SHA1 (Recipient-Public-Key)

The Recipient-Public-Key is passed to SHA1 with the MSB first. The exponent is assumed to be 3 and it not passed to SHA1.

3. Let Digest = SHA1 (Data-To-Seal).

4 Let Signature-Block = RSA-Private-Encrypt (Sender-Private-Key, Digest).

5. Let Sender-Cert-Chain be an array of bytes where the first byte is the number of certificates in the chain, and the remaining bytes are the concatenation of the certificates. Recall that certificates include length information, so the start of each certificate can be identified.

6. Let Data-To-Protect = Sender-Cert-Chain || Signature-Block || Data-To-Seal. Notice that the length of the Data-To-Seal is implied by the length of the record that contains this primitive.

7. Let CBC-Chain = 8 bytes of zero.

8. Perform SealEncryptedData (Data-Encryption-Key, CBC-Chain, Data-To-Protect, Protected-Data, Output-CBC-Chain)

9. Let Envelope-Body = Protected-Data.

10 Discard Output-CBC-Chain.

11 Protected-Data = Envelope-Recipient || Envelope-Block || Envelope-Body.

Notice that the RSA-Private and RSA-Public operations could be replaced with any asymmetric encryption system such as Elliptic Curve or NTRU. Notice also, that the order of the fields within blocks of data can be changed without effecting the security of this primitive. For example, the Protected-Data field could have the Envelope-Body block appearing first. Notice further, that the SHA1 function in step 2 (Let Envelope-Recipient = SHA1 (Recipient-Public-Key)) above can be replaced with any cryptographic digest function such as MD2, MD4, MD5, RIPEMD, RIPEMD-160, MD6, SHA-256, SHA-384, or SHA-512, by adjusting the size of the related data fields according to the output size of the digest function

Notice that the Data-Encryption-Key and the OAEP-Seed can be proper or improper subsets of each other. For example, the Data-Encryption-Key could be the first 128 bits of the OAEP-Seed, or the OAEP-Seed could be generated from the Data-Encryption-Key by adding a fixed padding or by adding bits that are a simple function (such as bit-selection or rolling-exclusive-or) of the Key.

1.4 StoryMail Secure Socket Layer

The LW SSL protocol runs on top of a reliable bi-directional byte stream such as TCP The byte stream is assumed to be insecure in the sense that bytes can be modified, recorded, replayed, inserted or deleted The protocol turns this byte stream into a record stream by sending blocks of information preceded by a header that identifies the type of the record and its length. Implementations of this protocol will want to organize the transmission of records to fall within a single IP packet that makes up the TCP byte stream The protocol assumes that the byte stream will deliver any bytes that are sent so there is no need to handle retransmissions or acknowledgements at the LW SSL layer (these are done at the TCP layer) The protocol does however detect deleted data If an application needs an

acknowledgement that some piece of data is received, it will do that at a higher layer (e.g., the StoryMail reader expects to fetch a story and will keep trying until it gets the whole story).

The protocol begins with a handshake phases that sends two records in each direction. The two records sent by the server can be combined into a single TCP/IP packet, so the total overhead is three packets. These records can be used to setup a new master key (MK) for parties that have not communicated with each other recently, or reuse an existing MK that is cached to improve performance (reducing computation overhead and communication bandwidth). At the end of this phase the parties will be mutually authenticate to each other.

After the handshake phase, the parties send data records that carry higher layer information such as a story message. They close the session using the normal TCP close mechanism. Notice that this means an attacker can close the TCP session as part of a denial of service attack. Such attacks are too hard to prevent to be worth preventing at this time.

Different keys are used by the client and server for sending data. This avoids possible replay attacks such as sending the client a message that it had originally sent to the server in order to trick the client into thinking that the message came from the server. The SSL protocol has this mechanism also.

1.4.1 Data Maintained by Each Party

The client and server maintain the following information

- Client Long Term State
  - Client's own RSA Private and Public Key Pair
  - Digital Certificate with Client's Public Key

    This is issued by StoryMail's CA, and is verifiable with the StoryMail root public key.
  - State of Pseudo Random Number Generator
- Client Per-Server State
  - Table of Server-Name and Master-Key values

    The KID for the MK is the hash of the MK itself, so there is no need to store it separately.
- Client Per-Session State
  - 128-bit Client-Write key
  - 64-bit CBC chain value for Client-Write
  - 128-bit Server-Write key
  - 64-bit CBC chain value for Server-Write
  - During session handshake the hash of Hello message that was sent
- Server Long Term State
  - Server's own RSA Private and Public Key Pair
  - Digital Certificate with Server's Public Key

    This is issued by StoryMail's CA, and is verifiable with the StoryMail root public key

- State of Pseudo Random Number Generator

- Server Per-Client State
  - Cache Table of KID and Master Key values The KID for the MK is the hash of the MK itself, but it is the index to this table, so it must be kept as a column. Rows can be deleted when they have not been used for some time or when space is needed.
  - Cache table of hash values for client certificates that have been validated. This table reduces the effort required to validate a client certificate.
- Server Per-Session State
  - 128-bit Client-Write key
  - 64-bit CBC chain value for Client-Write
  - 128-bit Server-Write key
  - 64-bit CBC chain value for Server-Write
  - During session handshake the hash of Hello and Accept message

1.4.2 Format of a Record

In a preferred embodiment, all of the StoryMail data items that are transmitted (called records as they are called in the SSL specification) have the same header format show below The header bytes are never encrypted, though they are included in cryptographic checksums.

- Type – 1 byte
- Version – 1 byte = 0 (high 4 bits reserved as extra length bits)
- Length – 2 bytes, MSB first = number of bytes in remaining content not including the four header bytes. If more than 65536 bytes are to be sent, then up to 4 bits of the version byte can be used to represent lengths up to 1 Mbyte The preferred way to send a large data item is to place it in several smaller records.
- Content bytes

1.4.3 Types of Records

The Type byte of a record can have the following meanings For the first release the version byte will be zero

- SM-Certificate = a certificate
- SM-Hello-New-MK = a new master key request
- SM-Accept-New-MK = response to new master key request
- SM-Hello-Reuse-MK = reuse master key request
- SM-Accept-Reuse-MK = response to reuse master key request
- SM-Reject-New-MK = negative response to reuse master key request